```
#include <stdio.h>
#include <stdlib.h>

typedef struct Node
{
   int val;
   struct Node *lchild;
   struct Node *rchild;
} Node;

Node *root = NULL;

void create();
void insert(int);
void inorder(Node *);
void preorder(Node *);
void postorder(Node *);

int main()
{
   int ch;
   root = NULL; // Initialize root to NULL
   do
   {
```

```
    printf("1.CREATE\t2.INORDER\t3.PREORDER\t4.POSTORDER\t5.EXIT\n");
    printf("Enter your choice\n");
    scanf("%d", &ch);
    switch (ch)
    {
    case 1:
       create();
       break;
    case 2:
       inorder(root);
       break;
    case 3:
       preorder(root);
       break;
    case 4:
       postorder(root);
       break;
    case 5:
       exit(0);
    default:
       printf("Invalid choice\n");
    }
} while (ch != 5);
return 0;
```

```
}

void create()
{
   int n, e;
   printf("Enter the number of elements\n");
   scanf("%d", &n);
   printf("Enter the elements one by one\n");
   for (int i = 0; i < n; i++)
   {
      scanf("%d", &e);
      insert(e);
   }
   printf("Tree constructed\n");
}

void insert(int e)
{
   Node *nn, *temp, *prev;
   nn = (Node *)malloc(sizeof(Node));
   nn->val = e;
   nn->lchild = NULL;
   nn->rchild = NULL;
   if (root == NULL)
```

```
    {
        root = nn;
        return;
    }
    temp = root;
    while (temp != NULL)
    {
        prev = temp;
        if (e < temp->val)
            temp = temp->lchild;
        else if (e > temp->val)
            temp = temp->rchild;
        else
        {
            printf("It's a duplicate node\n");
            free(nn); // Free the allocated memory for the duplicate node
            return;
        }
    }
    if (e < prev->val)
        prev->lchild = nn;
    else
        prev->rchild = nn;
}
```

```
void inorder(Node *tree)
{
   if (tree != NULL)
   {
      inorder(tree->lchild);
      printf("%d\n", tree->val);
      inorder(tree->rchild);
   }
}

void preorder(Node *tree)
{
   if (tree != NULL)
   {
      printf("%d\n", tree->val);
      preorder(tree->lchild);
      preorder(tree->rchild);
   }
}

void postorder(Node *tree)
{
   if (tree != NULL)
```

```
  {
    postorder(tree->lchild);
    postorder(tree->rchild);
    printf("%d\n", tree->val);
  }
}
```

```
1.CREATE        2.INORDER       3.PREORDER      4.POSTORDER     5.EXIT
Enter your choice
1
Enter the number of elements
3
Enter the elements one by one
2
3
1
Tree constructed
1.CREATE        2.INORDER       3.PREORDER      4.POSTORDER     5.EXIT
Enter your choice
2
1
2
3
1.CREATE        2.INORDER       3.PREORDER      4.POSTORDER     5.EXIT
Enter your choice
3
2
1
3
1.CREATE        2.INORDER       3.PREORDER      4.POSTORDER     5.EXIT
Enter your choice
4
1
3
2
1.CREATE        2.INORDER       3.PREORDER      4.POSTORDER     5.EXIT
Enter your choice
5
```